(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 1 184 790 A2

(12) **EUROPEAN PATENT APPLICATION**

(43) Date of publication:
06.03.2002 Bulletin 2002/10

(51) Int Cl.[7]: **G06F 11/36**

(21) Application number: **01203917.8**

(22) Date of filing: **07.04.1998**

(84) Designated Contracting States:
DE FR GB

(30) Priority: **08.04.1997 US 43070 P**
14.10.1997 US 949897

(62) Document number(s) of the earlier application(s) in accordance with Art. 76 EPC:
98915323.4 / 0 974 094

(71) Applicant: **ADVANCED MICRO DEVICES INC.**
Sunnyvale, California 94088-3453 (US)

(72) Inventor: **Mann, Daniel P.**
Austin, TX 78704 (US)

(74) Representative: **Brookes Batchellor**
102-108 Clerkenwell Road
London EC1M 5SA (GB)

Remarks:
This application was filed on 15 - 10 - 2001 as a divisional application to the application mentioned under INID code 62.

(54) **Trace cache for a microprocessor-based device**

(57) A processor-based device (102) incorporating an on-chip instruction trace cache (200) capable of providing information for reconstructing instruction execution flow. The trace information can be captured without halting normal processor (104) operation. Both serial (204) and parallel (214) communication channels are provided for communicating the trace information to external devices. In the disclosed embodiment of the invention, instructions that disrupt the instruction flow are reported, particularly instructions in which the target address is in some way data dependent. For example, call instructions or unconditional branch instructions in which the target address is provided from a data register (or other memory location such as a stack) cause a trace cache entry to be generated. In the case of many unconditional branches or sequential instructions, no entry is placed into the trace cache (200) because the target address can be completely determined from the instruction stream. Other information provided by the instruction trace cache (200) includes: the target address of a trap or interrupt handler, the target address of a return instruction, addresses from procedure returns, task identifiers, and trace capture stop/start information.

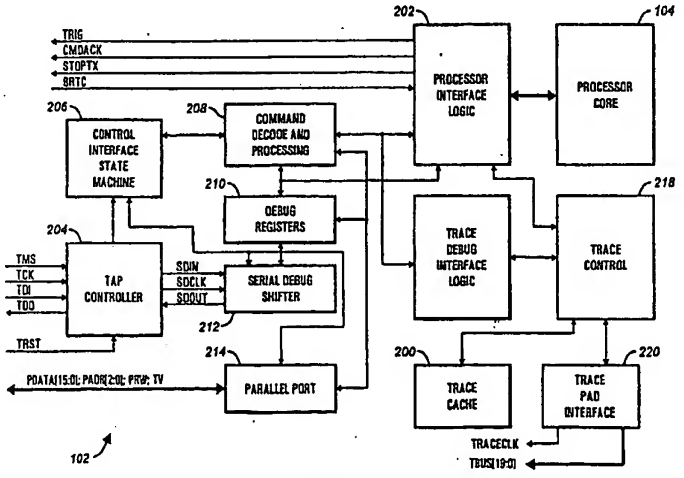


FIG. 2

EP 1 184 790 A2

Description

TECHNICAL FIELD

[0001] The invention relates to software debug support in microprocessors, and more particularly to a microprocessor-based device incorporating an on-chip instruction trace cache.

BACKGROUND ART

[0002] The growth in software complexity, coupled with increasing processor clock speeds, has placed an increasing burden on application software developers. The cost of developing and debugging new software products is now a significant factor in processor selection. A processor's failure to adequately facilitate software debug results in longer customer development times and reduces the processor's attractiveness for use within industry. The need to provide software debug support is particularly acute within the embedded products industry, where specialized on-chip circuitry is often combined with a processor core.

[0003] In addition to the software engineer, other parties are also affected by debug tool configuration. These parties include: the "trace" algorithm developer who must search through captured software trace data that reflects instruction execution flow in a processor; the in-circuit emulator developer who deals with problems of signal synchronization, clock frequency and trace bandwidth; and the processor manufacturer who does not want a solution that results in increased processor cost or design and development complexity.

[0004] With desktop systems, complex multitasking operating systems are currently available to support debugging. However, the initial task of getting these operating systems running reliably often requires special development equipment. While not the standard in the desktop environment, the use of such equipment is often the approach taken within the embedded industry. Logic analyzers, read-only memory (ROM) emulators and in-circuit emulators (ICE) are frequently employed. In-circuit emulators do provide certain advantages over other debug environments, offering complete control and visibility over memory and register contents, as well as overlay and trace memory in case system memory is insufficient. Use of traditional in-circuit emulators, which involves interfacing a custom emulator back-end with a processor socket to allow communication between emulation equipment and the target system, is becoming increasingly difficult and expensive in today's age of exotic packages and shrinking product life cycles.

[0005] Assuming full-function in-circuit emulation is required, there are a few known processor manufacturing techniques able to offer the required support for emulation equipment. Most processors intended for personal computer (PC) systems utilize a multiplexed approach in which existing pins are multiplexed for use in software debug. This approach is not particularly desirable in the embedded industry, where it is more difficult to overload pin functionality.

[0006] Other more advanced processors multiplex debug pins in time. In such processors, the address bus is used to report software trace information during a BTA-cycle (Branch Target Address). The BTA-cycle, however, must be stolen from the regular bus operation. In debug environments where branch activity is high and cache hit rates are low, it becomes impossible to hide the BTA-cycles. The resulting conflict over access to the address bus necessitates processor "throttle back" to prevent loss of instruction trace information. In the communications industry, for example, software typically makes extensive use of branching and suffers poor cache utilization, often resulting in 20% throttle back or more. This amount of throttling is unacceptable amount for embedded products which must accommodate real-time constrains.

[0007] In another approach, a second "trace" or "slave" processor is combined with the main processor, with the two processors operating in-step. Only the main processor is required to fetch instructions. The second, slave processor is used to monitor the fetched instructions on the data bus and keeps its internal state in synchronization with the main processor. The address bus of the slave processor functions to provide trace information. After power-up, via a JTAG (Joint Test Action Group) input, the second processor is switched into a slave mode of operation. Free from the need to fetch instructions, its address bus and other pins provide the necessary trace information.

[0008] Another existing approach involves building debug support into every processor, but only bonding-out the necessary signal pins in a limited number of packages. These "specially" packaged versions of the processor are used during debug and replaced with the smaller package for final production. This bond-out approach suffers from the need to support additional bond pad sites in all fabricated devices. This can be a burden in small packages and pad limited designs, particularly if a substantial number of "extra" pins are required by the debug support variant. Additionally, the debug capability ofthe specially packaged processors is unavailable in typical processor-based production systems.

[0009] In yet another approach (the "Background Debug Mode" by Motorola, Inc.) limited on-chip debug circuitry is provided for basic run control. Through a dedicated serial link requiring additional pins, this approach allows a debugger to start and stop the target system and apply basic code breakpoints by inserting special instructions in system memory. Once halted, special commands are used to inspect memory variables and register contents. This serial link, however, does not provide trace support -- additional dedicated pins and expensive external trace capture hardware are required

to provide instruction trace data. European patent application EP-A-0 762 276 of Motorola describes a debug module of a data processor which provides a parallel output port for providing internal operating information via a DDATA signal and a PST signal. The DDATA signal provides data which reflects operand values and the PST signal provides encoded status information which reflects an execution status of the central processing unit.

[0010] Thus, the current solutions for software debugging suffer from a variety of limitations, including: increased packaging and development costs, circuit complexity, processor throttling, and bandwidth matching difficulties. Further, there is currently no adequate low-cost procedure for providing trace information. The limitations of the existing solutions are likely to be exacerbated in the future as internal processor clock frequencies continue to increase.

DISCLOSURE OF THE INVENTION

[0011] Briefly, a processor-based device according to the present invention includes an on-chip instruction trace cache capable of providing information for reconstructing instruction execution flow. The trace information can be captured without halting normal processor operation. Both serial and parallel communication channels are provided for communicating the trace information to external devices. In the disclosed embodiment of the invention, controllability and observability of the instruction trace cache are achieved through a software debug port that uses an IEEE- 1 149.1-1990 compliant JTAG (Joint Test Action Group) interface or a similar standardized interface that is integrated into the processor-based device.

[0012] According to a first aspect, the present invention provides an electronic processor-based device adapted to execute a series of instructions obtained from external sources, the processor-based device being provided with pins to permit connection to external conductors, the electronic processor-based device being characterized by:

a trace cache coupled to a processor core for storing trace information indicative of the order in which the instructions are executed by the processor core, the trace cache comprising a series of storage elements, each storage element being adapted to store trace information, the trace information including a plurality of instruction trace records containing address and data information, the trace cache being configured to load data from the processor core in response to a load command and configured for the processor core to retrieve data from the trace cache in response to a retrieve command;

and a communication interface connected between the trace cache and selected ones of the pins to provide for transmission of trace information from the trace cache to external devices.

[0013] According to a second aspect the present invention provides method for analysing trace information in a processor-based device having a processor core comprising the steps of:

providing a trace cache within the processor-based device the trace cache comprising a series of storage elements adapted to store trace information;

capturing trace information from the processor core that is indicative of the order in which the series of instructions is executed by the processor core;

storing the trace information in the trace cache storage elements as instruction trace records;

retrieving the trace information by the processor core from the trace cache storage elements in response to a retrieve command; and

loading other information from the processor core into the trace cache storage elements in response to a load command;

providing a communication channel from the trace cache to selected pins of the processor-based device and communicating the trace information from the trace cache to the selected pins via the communication channel.

[0014] Preferably, information stored in the instruction trace cache is "compressed" such that a smaller cache can be utilized. In addition, compressing trace data allows external hardware to operate at normal bus speeds, even while the internal processor is operating much faster. Less expensive external capture hardware can therefore be utilized with a processor-based device according to the invention.

[0015] In the disclosed embodiment of the invention, if an address in an instruction stream can be obtained from a program image (Object Module), then it is not provided in the trace data. Preferably, only instructions that disrupt the instruction flow are reported; and further, only instructions in which the target address is in some way data dependent. Such "disrupting" events include, for example, call instructions or unconditional branch instructions in which the target address is provided from a data register or other memory location such as a stack. In the case of many unconditional branches or sequential instructions, no entry is placed into the trace cache because the target address can be completely determined from the instruction stream. Other information provided by the instruction trace cache includes: the target address of a trap or interrupt handler, the target address of a return instruction, addresses from procedure returns,

task identifiers, and trace capture stop/start information. This technique reduces the amount of information transferred from the trace cache to external debug hardware.

[0016] Thus, a processor-based device supplying a flexible, high-performance solution for furnishing instruction trace information is provided by the invention. The disclosed on-chip instruction trace cache alleviates various of the bandwidth and clock synchronization problems that arise in many existing solutions.

BRIEF DESCRIPTION OF DRAWINGS

[0017] A better understanding of the present invention can be obtained when the following detailed description of the preferred embodiment is considered in conjunction with the following drawings, in which:

Figure 1 is a block diagram of a software debug environment utilizing a software debug solution in accordance with the present invention;
Figure 2 is a block diagram providing details of an exemplary embedded processor product incorporating an on-chip instruction trace cache according to the present invention;
Figure 3 is a simplified block diagram depicting the relationship between an exemplary instruction trace cache and other components of an embedded processor product according to the present invention;
Figure 4 is a flowchart illustrating software debug command passing according to one embodiment of the invention;
Figure 5 is a flowchart illustrating enhanced software port command passing according to a second embodiment of the invention; and
Figures 6A - 6G illustrate the general format of a variety of trace cache entries for reporting instruction execution according to the invention.

MODE(S) FOR CARRYING OUT THE INVENTION

[0018] Turning now to the drawings, Figure 1 depicts an exemplary software debug environment illustrating a contemplated use of the present invention. A target system T is shown containing an embedded processor device 102 according to the present invention coupled to system memory 106. The embedded processor device 102 incorporates a processor core 104, an instruction trace cache 200 (Figure 2), and a debug port 100. Although not considered critical to the invention, the embedded processor device 102 may incorporate additional circuitry (not shown) for performing application specific functions, or may take the form of a stand-alone processor or digital signal processor. Preferably, the debug port 100 uses an IEEE-1149.1-1990 compliant JTAG interface or other similar standardized serial port interface.

[0019] A host system H is used to execute debug control software 112 for transferring high-level commands and controlling the extraction and analysis of debug information generated by the target system T. The host system H and target system T of the disclosed embodiment of the invention communicate via a serial link 110. Most computers are equipped with a serial or parallel interface which can be inexpensively connected to the debug port 100 by means of a serial connector 108, allowing a variety of computers to function as a host system H. Alternatively, the serial connector 108 could be replaced with higher speed JTAG-to-network conversion equipment. Further, the target system T can be configured to analyze debug/trace information internally.

[0020] Referring now to Figure 2, details of an embedded processor device 102 according to the present invention are provided. In addition to the processor core 104, Figure 2 depicts various elements of an enhanced embodiment of the debug port 100 capable of utilizing and controlling the trace cache 200. Many other configurations are possible, as will become apparent to those skilled in the art, and the various processor device 102 components described below are shown for purposes of illustrating the benefits associated with providing an on-chip trace cache 200.

[0021] Of significance to the disclosed embodiment of the invention, the trace control circuitry 218 and trace cache 200 operate to provide trace information for reconstructing instruction execution flow in the processor core 104. The trace control circuitry 218 supports "tracing" to a trace pad interface port 220 or to the instruction trace cache 200 and provides user control for selectively activating instruction trace capture. Other features enabled by the trace control circuitry 218 include programmability of synchronization address generation and user specified trace records, as discussed in greater detail below. The trace control circuitry 218 also controls a trace pad interface port 220. When utilized, the trace pad interface port 220 is capable of providing trace data while the processor core 104 is executing instructions, although clock synchronization and other issues may arise. The instruction trace cache 200 addresses many of these issues, improving bandwidth matching and alleviating the need to incorporate throttle-back circuitry in the processor core 104.

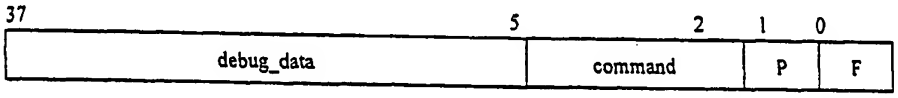
[0022] At a minimum, only the conventional JTAG pins need be supported in the software debug port 100 in the described embodiment of the invention. The JTAG pins essentially become a transportation mechanism, using existing pins, to enter commands to be performed by the processor core 104. More specifically, the test clock signal TCK, the

test mode select signal TMS, the test data input signal TDI and the test data output signal TDO provided to and driven by the JTAG Test Access Port (TAP) controller 204 are conventional JTAG support signals and known to those skilled in the art. As discussed in more detail below, an "enhanced" embodiment of the debug port 100 adds the command acknowledge signal CMDACK, the break request/trace capture signal BRTC, the stop transmit signal STOPTX, and the trigger signal TRIG to the standard JTAG interface. The additional signals allow for pinpoint accuracy of external breakpoint assertion and monitoring, triggering of external devices in response to internal breakpoints, and elimination of status polling of the JTAG serial interface. These "sideband" signals offer extra functionality and improve communications speeds for the debug port 100. These signals also aid in the operation of an optional parallel port 214 provided on special bond-out versions of the disclosed embedded processor device 102.

[0023] Via the conventional JTAG signals, the JTAG TAP controller 204 accepts standard JTAG serial data and control. When a DEBUG instruction has been written to the JTAG instruction register, a serial debug shifter 212 is connected to the JTAG test data input signal TDI and test data output signal TDO, such that commands and data can then be loaded into and read from debug registers 210. In the disclosed embodiment of the invention, the debug registers 210 include two debug registers for transmitting (TX_DATA register) and receiving (RX_DATA register) data, an instruction trace configuration register (ITCR), and a debug control status register (DCSR).

[0024] A control interface state machine 206 coordinates the loading/reading of data to/from the serial debug shifter 212 and the debug registers 210. A command decode and processing block 208 decodes commands/data and dispatches them to processor interface logic 202 and trace debug interface logic 216. In addition to performing other functions, the trace debug interface logic 216 and trace control logic 218 coordinate the communication of trace information from the trace cache 200 to the TAP controller 204. The processor interface logic 202 communicates directly with the processor core 104, as well as the trace control logic 218. As described more fully below, parallel port logic 214 communicates with a control interface state machine 206 and the debug registers 210 to perform parallel data read/write operations in optional bond-out versions of the embedded processor device 102.

[0025] Before debug information is communicated via the debug port 100 using only conventional JTAG signals, the port 100 is enabled by writing the public JTAG instruction DEBUG into a JTAG instruction register contained within the TAP controller 204. As shown below, the JTAG instruction register of the disclosed embodiment is a 38-bit register comprising a 32-bit data field (debug_data[31:0]), a four-bit command field to point to various internal registers and functions provided by the debug port 100, a command pending flag, and a command finished flag. It is possible for some commands to use bits from the debug_data field as a sub-field to extend the number of available commands.



JTAG Instruction Register.

[0026] This JTAG instruction register is selected by toggling the test mode select signal TMS. The test mode select signal TMS allows the JTAG path of clocking to be changed in the scan path, enabling multiple paths of varying lengths to be used. Preferably, the JTAG instruction register is accessible via a short path. This register is configured to include a "soft" register for holding values to be loaded into or received from specified system registers.

[0027] Referring now to Figure 3, a simplified block diagram depicting the relationship between an exemplary instruction trace cache 200 and other components of an embedded processor device 102 according to the present invention is shown. In one contemplated embodiment of the invention, the trace cache 200 is a 128 entry first-in, first-out (FIFO) circular cache that records the most recent trace entries. Increasing the size of the trace cache 200 increases the amount of instruction trace information that can be captured, although the amount of required silicon area may increase.

[0028] As described in more detail below, the trace cache 200 of the disclosed embodiment of the invention stores a plurality of 20-bit (or more) trace entries indicative of the order in which instructions are executed by the processor core 104. Other information, such as task identifiers and trace capture stop/start information, can also be placed in the trace cache 200. The contents of the trace cache 200 are provided to external hardware, such as the host system H, via either serial or parallel trace pins 230. Alternatively, the target system T can be configured to examine the contents of the trace cache 200 internally.

[0029] Figure 4 provides a high-level flow chart of command passing when using a standard JTAG interface. Upon entering debug mode in step 400 the DEBUG instruction is written to the TAP controller 204 in step 402. Next, step 404, the 38-bit serial value is shifted in as a whole, with the command pending flag set and desired data (if applicable, otherwise zero) in the data field. Control proceeds to step 406 where the pending command is loaded/unloaded and

the command finished flag checked. Completion of a command typically involves transferring a value between a data register and a processor register or memory/IO location. After the command has been completed, the processor 104 clears the command pending flag and sets the command finished flag, at the same time storing a value in the data field if applicable. The entire 38-bit register is scanned to monitor the command finished and command pending flags. If the pending flag is reset to zero and the finished flag is set to one, the previous command has finished. The status of the flags is captured by the control interface state machine 206. A slave copy of the flags' status is saved internally to determine if the next instruction should be loaded. The slave copy is maintained due to the possibility of a change in flag status between TAP controller 204 states. This allows the processor 104 to determine if the previous instruction has finished before loading the next instruction.

[0030] If the finished flag is not set as determined in step 408, control proceeds to step 410 and the loading/unloading of the 38-bit command is repeated. The command finished flag is also checked. Control then returns to step 408. If the finished flag is set as determined in step 408, control returns to step 406 for processing of the next command. DEBUG mode is exited via a typical JTAG process.

[0031] Returning to Figure 2, the aforementioned optional sideband signals are utilized in the enhanced debug port 100 to provide extra functionality. The optional sideband signals include a break request/trace capture signal BRTC that can function as a break request signal or a trace capture enable signal depending on the status of bit set in the debug control/status register. If the break request/trace capture signal BRTC is set to function as a break request signal, it is asserted to cause the processor 104 to enter debug mode (the processor 104 can also be stopped by scanning in a halt command via the convention JTAG signals). If set to function as a trace capture enable signal, asserting the break request/trace capture signal BRTC enables trace capture. Deasserting the signal turns trace capture off. The signal takes effect on the next instruction boundary after it is detected and is synchronized with the internal processor clock. The break request/trace capture signal BRTC may be asserted at any time.

[0032] The trigger signal TRIG is configured to pulse whenever an internal processor breakpoint has been asserted. The trigger signal TRIG may be used to trigger an external capturing device such as a logic analyzer, and is synchronized with the trace record capture clock signal TRACECLK. When a breakpoint is generated, the event is synchronized with the trace capture clock signal TRACECLK, after which the trigger signal TRIG is held active for the duration of trace capture.

[0033] The stop transmit signal STOPTX is asserted when the processor 104 has entered DEBUG mode and is ready for register interrogation/modification, memory or I/O reads and writes through the debug port 100. In the disclosed embodiment of the invention, the stop transmit signal STOPTX reflects the state of a bit in the debug control status register (DCSR). The stop transmit signal STOPTX is synchronous with the trace capture clock signal TRACECLK.

[0034] The command acknowledge signal CMDACK is described in conjunction with Figure 5, which shows simplified command passing in the enhanced debug port 100 of Figure 2. Again, to place the target system T into DEBUG mode, a DEBUG instruction is written to the TAP controller 204 in step 502. Control proceeds to step 504 and the command acknowledge signal CMDACK is monitored by the host system H to determine command completion status. This signal is asserted high by the target system T simultaneously with the command finished flag and remains high until the next shift cycle begins. When using the command acknowledge signal CMDACK, it is not necessary to shift out the JTAG instruction register to capture the command finished flag status. The command acknowledge signal CMDACK transitions high on the next rising edge of the test clock signal TCK after the command finished flag has changed from zero to one. When using the enhanced JTAG signals, a new shift sequence (step 506) is not started by the host system H until the command acknowledge signal CMDACK pin has been asserted high. The command acknowledge signal CMDACK is synchronous with the test clock signal TCK. The test clock signal TCK need not be clocked at all times, but is ideally clocked continuously when waiting for a command acknowledge signal CMDACK response.

OPERATING SYSTEM/APPLICATION COMMUNICATION VIA THE DEBUG PORT 100

[0035] Also included in debug register block 210 is an instruction trace configuration register (ITCR). This 32-bit register provides for the enabling/disabling and configuration of instruction trace debug functions. Numerous such functions are contemplated, including various levels of tracing, trace synchronization force counts, trace initialization, instruction tracing modes, clock divider ratio information, as well as additional functions shown in the following table. The ITCR is accessed through a JTAG instruction register write/read command as is the case with the other registers of the debug register block 210, or via a reserved instruction.

Instruction Trace Configuration Register (ITCR).		
BIT	SYMBOL	DESCRIPTION/FUNCTION
31:30	Reserved	Reserved
29	RXINTEN	Enables interrupt when RX bit is set
28	TXINTEN	Enables interrupt when TX bit is set
27	TX	Indicates that the target system T is ready to transmit data to the host system H and the data is available in the TX_DATA register
26	RX	Indicates that data has been received from the host and placed in the RX_DATA register
25	DISL1TR	Disables level 1 tracing
24	DISL0TR	Disables level 0 tracing
23	DISCSB	Disables current segment base trace record
22:16	TSYNC[6:0]	Sets the maximum number of Branch Sequence trace records that may be output by the trace control block 218 before a synchronizing address record is forced
15	TSR3	Sets or clears trace mode on DR3 trap
14	TSR2	Sets or clears trace mode on DR2 trap
13	TSR1	Sets or clears trace mode on DR1 trap
12	TSR0	Sets or clears trace mode on DR0 trap
11	TRACE3	Enables Trace mode toggling using DR3
10	TRACE2	Enables Trace mode toggling using DR2
9	TRACE1	Enables Trace mode toggling using DR1
8	TRACE0	Enables Trace mode toggling using DR0
7	TRON	Trace on/off
6:4	TCLK[2:0]	Encoded divider ratio between internal processor clock and TRACECLK
3	ITM	Sets internal or external (bond-out) instruction tracing mode
2	TINIT	Trace initialization
1	TRIGEN	Enables pulsing of external trigger signal TRIG following receipt of any legacy debug breakpoint; independent of the Debug Trap Enable function in the DCSR
0	GTEN	Global enable for instruction tracing through the internal trace buffer or via the external (bond-out) interface

[0036] Another debug register, the debug control/status register (DCSR), provides an indication of when the processor 104 has entered debug mode and allows the processor 104 to be forced into DEBUG mode through the enhanced JTAG interface. As shown in the following table, the DCSR also enables miscellaneous control features, such as: forcing a ready signal to the processor 104, controlling memory access space for accesses initiated through the debug port, disabling cache flush on entry to the DEBUG mode, the TX and RX bits, the parallel port 214 enable, forced breaks, forced global reset, and other functions. The ordering or presence of the various bits in either the ITCR or DCSR is not considered critical to the operation of the invention.

Debug Control/Status Register (DCSR).		
BIT	SYMBOL	DESCRIPTION/FUNCTION
31:12	Reserved	Reserved
11	TX	Indicates that the target system T is ready to transmit data to the host system H and the data is available in the TX_DATA register

(continued)

Debug Control/Status Register (DCSR).		
BIT	SYMBOL	DESCRIPTION/FUNCTION
10	RX	Indicates that data has been received from the host and placed in the RX_DATA register
9	DISFLUSH	Disables cache flush on entry to DEBUG mode
8	SMMSP	Controls memory access space (normal memory space/ system management mode memory) for accesses initiated through the Debug Port 100
7	STOP	Indicates whether the processor 104 is in DEBUG mode (equivalent to stop transmit signal STOPTX
6	FRCRDY	Forces the ready signal RDY to the processor 104 to be pulsed for one processor clock; useful when it is apparent that the processor 104 is stalled waiting for a ready signal from a non-responding device
5	BRKMODE	Selects the function of the break request/trace capture signal BRTC (break request or trace capture on/off)
4	DBTEN	Enables entry to debug mode or toggle trace mode enable on a trap/fault via processor 104 registers DR0-DR7 or other legacy debug trap/fault mechanisms
3	PARENB	Enables parallel port 214
2	DSPC	Disables stopping of internal processor clocks in the Halt and Stop Grant states
1	FBRK	Forces processor 104 into DEBUG mode at the next instruction boundary (equivalent to pulsing the external BRTC pin)
0	FRESET	Forces global reset

[0037] When in cross debug environment such as that of Figure 1, it is necessary for the parent task running on the target system T to send information to the host platform H controlling it. This data may consist, for example, of a character stream from a printf() call or register information from a Task's Control Block (TCB). One contemplated method for transferring the data is for the operating system to place the data in a known region, then via a trap instruction, cause DEBUG mode to be entered.

[0038] Via debug port 100 commands, the host system H can then determine the reason that DEBUG mode was entered, and respond by retrieving the data from the reserved region. However, while the processor 104 is in DEBUG mode, normal processor execution is stopped. As noted above, this is undesirable for many real-time systems.

[0039] This situation is addressed according to the present invention by providing two debug registers in the debug port 100 for transmitting (TX_DATA register) and receiving (RX_DATA register) data. These registers can be accessed using the soft address and JTAG instruction register commands. As noted, after the host system H has written a debug instruction to the JTAG instruction register, the serial debug shifter 212 is coupled to the test data input signal TDI line and test data output signal TDO line.

[0040] When the processor 104 executes code causing it to transmit data, it first tests a TX bit in the ITCR. If the TX bit is set to zero then the processor 104 executes a processor instruction (either a memory or I/O write) to transfer the data to the TX_DATA register. The debug port 100 sets the TX bit in the DCSR and ITCR, indicating to the host system H that it is ready to transmit data. Also, the STOPTX pin is set high. After the host system H completes reading the transmit data from the TX_DATA register, the TX bit is set to zero. A TXINTEN bit in the ITCR is then set to generate a signal to interrupt the processor 104. The interrupt is generated only when the TX bit in the ITCR transitions to zero. When the TXINTEN bit is not set, the processor 104 polls the ITCR to determine the status of the TX bit to further transmit data.

[0041] When the host system H desires to send data, it first tests a RX bit in the ITCR. If the RX bit is set to zero, the host system H writes the data to the RX_DATA register and the RX bit is set to one in both the DCSR and ITCR. A RXINT bit is then set in the ITCR to generate a signal to interrupt the processor 104. This interrupt is only generated when the RX in the ITCR transitions to one. When the RXINTEN bit is not set, the processor 104 polls the ITCR to verify the status of the RX bit. If the RX bit is set to one, the processor instruction is executed to read data from the RX_DATA register. After the data is read by the processor 104 from the RX_DATA register the RX bit is set to zero. The host system H continuously reads the ITCR to determine the status of the RX bit to further send data.

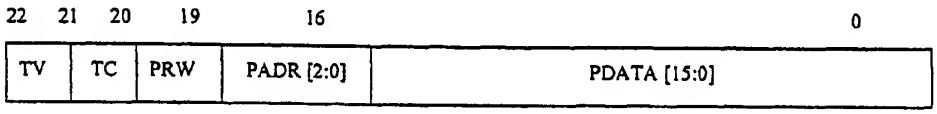
[0042] This technique enables an operating system or application to communicate with the host system H without

stopping processor 104 execution. Communication is conveniently achieved via the debug port 100 with minimal impact to on-chip application resources. In some cases it is necessary to disable system interrupts. This requires that the RX and TX bits be examined by the processor 100. In this situation, the communication link is driven in a polled mode.

PARALLEL INTERFACE TO DEBUG PORT 100

[0043] Some embedded systems require instruction trace to be examined while maintaining I/O and data processing operations. Without the use of a multi-tasking operating system, a bond-out version of the embedded processor device 102 is preferable to provide the trace data, as examining the trace cache 200 via the debug port 100 requires the processor 104 to be stopped.

[0044] In the disclosed embodiment of the invention, a parallel port 214 is also provided in an optional bond-out version of the embedded processor device 102 to provide parallel command and data access to the debug port 100. This interface provides a 16-bit data path that is multiplexed with the trace pad interface port 220. More specifically, the parallel port 214 provides a 16-bit wide bi-directional data bus (PDATA[15:0]), a 3-bit address bus (PADR[2:0]), a parallel debug port read/write select signal (PRW), a trace valid signal TV and an instruction trace record output clock TRACECLOCK (TC). Although not shared with the trace pad interface port 220, a parallel bus request/grant signal pair PBREQ/PBGNT (not shown) are also provided. The parallel port 214 is enabled by setting a bit in the DCSR. Serial communications via the debug port 100 are not disabled when the parallel port 214 is enabled.



Bond-Out Pins/Parallel Port 214 Format.

[0045] The parallel port 214 is primarily intended for fast downloads/uploads to and from target system T memory. However, the parallel port 214 may be used for all debug communications with the target system T whenever the processor 104 is stopped. The serial debug signals (standard or enhanced) are used for debug access to the target system T when the processor 104 is executing instructions.

[0046] In a similar manner to the JTAG standard, all inputs to the parallel port 214 are sampled on the rising edge of the test clock signal TCK, and all outputs are changed on the falling edge of the test clock signal TCK. In the disclosed embodiment, the parallel port 214 shares pins with the trace pad interface 220, requiring parallel commands to be initiated only while the processor 104 is stopped and the trace pad interface 220 is disconnected from the shared bus.

[0047] The parallel bus request signal PBREQ and parallel bus grant signal PBGNT are provided to expedite multiplexing of the shared bus signals between the trace cache 200 and the parallel port 214. When the host interface to the parallel port 214 determines that the parallel bus request signal PBREQ is asserted, it begins driving the parallel port 214 signals and asserts the parallel bus grant signal PBGNT.

[0048] When entering or leaving DEBUG mode with the parallel port 214 enabled, the parallel port 214 is used for the processor state save and restore cycles. The parallel bus request signal PBREQ is asserted immediately before the beginning of a save state sequence penultimate to entry of DEBUG mode. On the last restore state cycle, the parallel bus request signal PBREQ is deasserted after latching the write data. The parallel port 214 host interface responds to parallel bus request signal PBREQ deassertion by tri-stating its parallel port drivers and deasserting the parallel bus grant signal PBGNT. The parallel port 214 then enables the debug trace port pin drivers, completes the last restore state cycle, asserts the command acknowledge signal CMDACK, and returns control of the interface to trace control logic 218.

[0049] When communicating via the parallel port 214, the address pins PADR[2:0] are used for selection of the field of the JTAG instruction register, which is mapped to the 16-bit data bus PDATA[15:0] as shown in the following table:

PADR[2:0]	Data Selection
0 0 0	No selection (null operation)
0 0 1	4-bit command register; command driven on PDATA[3:0]
0 1 0	High 16-bits of debug_data
0 1 1	Low 16-bits of debug_data

(continued)

PADR[2:0]	Data Selection
100-111	Reserved

[0050] It is not necessary to update both halves of the debug_data [31:0] register if only one of the halves is being used (e.g., on 8-bit I/O cycle data writes). The command pending flag is automatically set when performing a write operation to the four-bit command register, and is cleared when the command finished flag is asserted. The host system H can monitor the command acknowledge signal CMDACK to determine when the finished flag has been asserted. Use of the parallel port 214 provides full visibility of execution history, without requiring throttling of the processor core 104. The trace cache 200, if needed, can be configured for use as a buffer to the parallel port 214 to alleviate any bandwidth matching issues.

OPERATING SYSTEM AND DEBUGGER INTEGRATION

[0051] In the disclosed embodiment of the invention, the operation of all debug supporting features, including the trace cache 200, can be controlled through the debug port 100 or via processor instructions. These processor instructions may be from a monitor program, target hosted debugger, or conventional pod-wear. The debug port 100 performs data moves which are initiated by serial data port commands rather than processor instructions.

[0052] Operation of the processor from conventional pod-space is very similar to operating in DEBUG mode from a monitor program. All debug operations can be controlled via processor instructions. It makes no difference whether these instructions come from pod-space or regular memory. This enables an operating system to be extended to include additional debug capabilities.

[0053] Of course, via privileged system calls such a ptrace(), operating systems have long supported debuggers. However, the incorporation of an on-chip trace cache 200 now enables an operating system to offer instruction trace capability. The ability to trace is often considered essential in real-time applications. In a debug environment according to the present invention, it is possible to enhance an operating system to support limited trace without the incorporation of an "external" logic analyzer or in-circuit emulator.

[0054] Examples of instructions used to support internal loading and retrieving of trace cache 200 contents include a load instruction trace cache record command LITCR and a store instruction trace cache record command SITCR. The command LITCR loads an indexed record in the trace cache 200, as specified by a trace cache pointer ITREC. PTR, with the contents of the EAX register of the processor core 104. The trace cache pointer ITREC.PTR is pre-incremented, such that the general operation of the command LITCR is as follows:

ITREC.PTR <- ITREC.PTR +1;
ITREC*[ITREC.PTR]* <- EAX.

In the event that the instruction trace record (see description of trace record format below) is smaller that the EAX record, only a portion of the EAX register is utilized.

[0055] Similarly, the store instruction trace cache record command SITCR is used to retrieve and store (in the EAX register) an indexed record from the trace cache 200. The contents of the ECX register of the processor core 104 are used as an offset that is added to the trace cache pointer ITREC.PTR to create an index into the trace cache 200. The ECX register is post-incremented while the trace cache pointer ITREC.PTR is unaffected, such that:

EAX <- ITREC*[ECX + ITREC.PTR]*;
ECX <- ECX + 1.

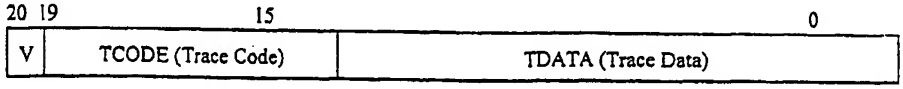
Numerous variations to the format of the LITCR and SITCR commands will be evident to those skilled art.

[0056] Extending an operating system to support on-chip trace has certain advantages within the communications industry. It enables the system I/O and communication activity to be maintained while a task is being traced. Traditionally, the use of an in-circuit emulator has necessitated that the processor be stopped before the processor's state and trace can be examined [unlike ptrace()]. This disrupts continuous support of I/O data processing.

[0057] Additionally, the trace cache 200 is very useful when used with equipment in the field. If an unexpected system crash occurs, the trace cache 200 can be examined to observe the execution history leading up to the crash event When used in portable systems or other environments in which power consumption is a concern, the trace cache 200 can be disabled as necessary via power management circuitry.

EXEMPLARY TRACE RECORD FORMAT

[0058] In the disclosed embodiment of the invention, an instruction trace record is 20 bits wide and consists of two fields, TCODE (Trace Code) and TDATA (Trace Data), as well as a valid bit V. The TCODE field is a code that identifies the type of data in the TDATA field. The TDATA field contains software trace information used for debug purposes.



Instruction Trace Record Format.

[0059] In one contemplated embodiment of the invention, the embedded processor device 102 reports eleven different trace codes as set forth in the following table:

TCODE#	TCODE Type	TDATA
0000	Missed Trace	Not Valid
0001	Conditional Branch	Contains Branch Sequence
0010	Branch Target	Contains Branch Target Address
0011	Previous Segment Base	Contains Previous Segment Base Address and Attributes
0100	Current Segment Base	Contains Current Segment Base Address and Attributes
0101	Interrupt	Contains Vector Number of Exception or Interrupt
0110	Trace Synchronization	Contains Address of Most Recently Executed Instruction
0111	Multiple Trace	Contains 2nd or 3rd Record of Entry With Multiple Records
1000	Trace Stop	Contains Instruction Address Where Trace Capture Was Stopped
1001	User Trace	Contains User Specified Trace Data
1010	Performance Profile	Contains Performance Profiling Data

[0060] The trace cache 200 is of limited storage capacity; thus a certain amount of "compression" in captured trace data is desirable. In capturing trace data, the following discussion assumes that an image of the program being traced is available to the host system H. If an address can be obtained from a program image (Object Module), then it is not provided in the trace data. Preferably, only instructions which disrupt the instruction flow are reported; and further, only those where the target address is in some way data dependent. For example, such "disrupting" events include call instructions or unconditional branch instructions in which the target address is provided from a data register or other memory location such as a stack.

[0061] As indicated in the preceding table, other desired trace information includes: the target address of a trap or interrupt handler; the target address of a return instruction; a conditional branch instruction having a target address which is data register dependent (otherwise, all that is needed is a 1-bit trace indicating if the branch was taken or not); and, most frequently, addresses from procedure returns. Other information, such as task identifiers and trace capture stop/start information, can also be placed in the trace cache 200. The precise contents and nature of the trace records are not considered critical to the invention.

[0062] Figure 6A illustrates an exemplary format for reporting conditional branch events. In the disclosed embodiment of the invention, the outcome of up to 15 branch events can be grouped into a single trace entry. The 16-bit TDATA field (or "BFIELD") contains 1-bit branch outcome trace entries, and is labeled as a TCODE = 0001 entry. The TDATA field is initially cleared except for the left most bit, which is set to 1. As each new conditional branch is encountered, a new one bit entry is added on the left and any other entries are shifted to the right by one bit.

[0063] Using a 128 entry trace cache 200 allows 320 bytes of information to be stored. Assuming a branch frequency of one branch every six instructions, the disclosed trace cache 200 therefore provides an effective trace record of 1,536 instructions. This estimate does not take into account the occurrence of call, jump and return instructions.

[0064] In the disclosed embodiment of the invention, the trace control logic 218 monitors instruction execution via

processor interface logic 202. When a branch target address must be reported, information contained within a current conditional branch TDATA field is marked as complete by the trace control logic 218, even if 15 entries have not accumulated. As shown in Figure 6B, the target address (in a processor-based device 102 using 32-bit addressing) is then recorded in a trace entry pair, with the first entry (TCODE = 0010) providing the high 16-bits of the target address and the second entry (TCODE = 0111) providing the low 16-bits of the target address. When a branch target address is provided for a conditional jump instruction, no 1-bit branch outcome trace entry appears for the reported branch.

STARTING AND STOPPING TRACE CAPTURE

[0065] Referring now to Figure 6C, it may be desirable to start and stop trace gathering during certain sections of program execution; for example, when a task context switch occurs. When trace capture is stopped, no trace entries are entered into the trace cache 200, nor do any appear on the bond-out pins of trace port 214. Different methods are contemplated for enabling and disabling trace capture. For example, an x86 command can be provided, or an existing x86 command can be utilized to toggle a bit in an I/O port location. Alternatively, on-chip breakpoint control registers (not shown) can be configured to indicate the addresses where trace capture should start/stop. When tracing is halted, a trace entry (TCODE = 1000, TCODE = 0111) recording the last trace address is placed in the trace stream. When tracing is resumed, a trace synchronization entry (TCODE = 0110, TCODE = 0111) containing the address of the currently executing instruction is generated.

[0066] It may be important to account for segment changes that occur while tracing is stopped. This situation can be partially resolved by selecting an option to immediately follow a TCODE = 1000 entry with a current segment base address entry (TCODE = 0100, TCODE = 0111), as shown in Figure 6C. A configuration option is also desirable to enable a current segment base address entry at the end of a trace prior to entering Debug mode. By contrast, it may not be desirable to provide segment base information when the base has not changed, such as when an interrupt has occurred.

[0067] Referring to Figure 6D, following the occurrence of an asynchronous or synchronous event such as an interrupt or trap, a TCODE = 0101 trace entry is generated to provide the address of the target interrupt handler. However, it is also desirable to record the address of the instruction which was interrupted by generating a trace synchronization (TCODE = 0110) entry immediately prior to the interrupt entry, as well as the previous segment base address (TCODE = 0011). The trace synchronization entry contains the address of the last instruction retired before the interrupt handler commences.

SEGMENT CHANGES

[0068] Figure 6E illustrates a trace entry used to report a change in segment parameters. When processing a trace stream in accordance with the invention, trace address values are combined with a segment base address to determine an instruction's linear address. The base address, as well as the default data operand size (32 or 16-bit mode), are subject to change. As a result, the TCODE = 0011 and 0111 entries are configured to provide the information necessary to accurately reconstruct instruction flow. The TDATA field corresponding to a TCODE = 0011 entry contains the high 16-bits of the previous segment base address, while the associated TCODE = 0111 entry contains the low 15 or 4 bits (depending on whether the instruction is executed in real or protected mode). The TCODE = 0111 entry also preferably includes bits indicating the current segment size (32-bit or 16-bit), the operating mode (real or protected), and a bit indicating whether paging is being utilized. Segment information generally relates to the previous segment, not a current (target) segment. Current segment information is obtained by stopping and examining the state of the processor core 104.

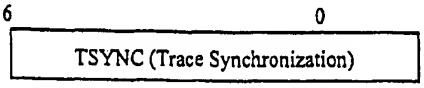
USER SPECIFIED TRACE ENTRY

[0069] There are circumstance when an application program or operating system may wish to add additional information into a trace stream. For this to occur, an x86 instruction is preferably provided which enables a 16-bit data value to be placed in the trace stream at a desired execution position. The instruction can be implemented as a move to I/O space, with the operand being provided by memory or a register. When the processor core 104 executes this instruction, the user specified trace entry is captured by the trace control logic 218 and placed in the trace cache 200. As shown in Figure 6F, a TCODE = 1001 entry is used for this purpose in the disclosed embodiment of the invention. This entry might provide, for example, a previous or current task identifier when a task switch occurs in a multi-tasking operating system.

SYNCHRONIZATION OF TRACE DATA

[0070] When executing typical software on a processor-based device 102 according to the disclosed embodiment of the invention, few trace entries contain address values. Most entries are of the TCODE = 0001 format, in which a single bit indicates the result of a conditional operation. When examining a trace stream, however, data can only be studied in relation to a known program address. For example, starting with the oldest entry in the trace cache 200, all entries until an address entry are of little use. Algorithm synchronization typically begins from a trace entry providing a target address.

[0071] If the trace cache 200 contains no entries providing an address, then trace analysis cannot occur. This situation is rare, but possible. For this reason, a synchronization register TSYNC is provided in the preferred embodiment of invention to control the injection of synchronizing address information. If the synchronization register TSYNC is set to zero, then trace synchronization entries are not generated.



Trace Entry Synchronization Entry Control Register.

[0072] Figure 6G depicts an exemplary trace synchronization entry. In operation, a counter register is set to the value contained in the synchronization register TSYNC whenever a trace entry containing a target address is generated. The counter is decremented by one for all other trace entries. If the counter reaches zero, a trace entry is inserted (TCODE = 0110) containing the address of the most recently retired instruction (or, alternatively, the pending instruction). In addition, when a synchronizing entry is recorded in the trace cache 200, it also appears on the trace pins 220 to ensure sufficient availability of synchronizing trace data for full-function ICE equipment.

[0073] Trace entry information can also be expanded to include data relating to code coverage or execution performance. This information is useful, for example, for code testing and performance tuning. Even without these enhancements, it is desirable to enable the processor core 104 to access the trace cache 200. In the case of a microcontroller device, this feature can be accomplished by mapping the trace cache 200 within a portion of I/O or memory space. A more general approach involves including an instruction which supports moving trace cache 200 data into system memory.

[0074] Thus, a processor-based device providing a flexible, high-performance solution for furnishing instruction trace information has been described. The processor-based device incorporates an instruction trace cache capable of providing trace information for reconstructing instruction execution flow on the processor without halting processor operation. Both serial and parallel communication channels are provided for communicating trace data to external devices. The disclosed on-chip instruction trace cache alleviates various of the bandwidth and clock synchronization problems that arise in many existing solutions, and also allows less expensive external capture hardware to be utilized.

[0075] The foregoing disclosure and description of the invention are illustrative and explanatory thereof, and various changes in the size, shape, materials, components, circuit elements, wiring connections and contacts, as well as in the details of the illustrated circuitry and construction and method of operation may be made without departing from the spirit of the invention.

Claims

1. An electronic processor-based device (102) adapted to execute a series of instructions obtained from external sources (106), the processor-based device being provided with pins to permit connection to external conductors, the electronic processor-based device being **characterized by**:

   a trace cache (200) coupled to a processor core (104) for storing trace information indicative of the order in which the instructions are executed by the processor core, the trace cache comprising a series of storage elements, each storage element being adapted to store trace information, the trace information including a plurality of instruction trace records containing address and data information, the trace cache (200) being configured to load data from the processor core (104) in response to a load command and configured for the processor core (104) to retrieve data from the trace cache (200) in response to a retrieve command;

and a communication channel connected between the trace cache (200) and selected ones of the pins to provide for transmission of trace information from the trace cache to external devices.

2. The processor-based device of claim 1 wherein the instruction trace records each include a trace data field (TDATA), and a trace code field (TCODE) for storing a code to identify the type of data in the trace data field.

3. The processor-based device of claim 1 or 2 configured to omit from stored trace information instruction trace records for instructions between instructions that disrupt the instruction flow.

4. The processor-based device of claim 1, 2 or 3 wherein information concerning executed branch instructions having a target address that is not data register dependent is stored in the trace cache in the form of a single bit.

5. The processor-based device of any preceding claim, wherein the trace cache (200) is further configured to provide trace capture start/stop information.

6. The processor-based device of any preceding claim, wherein the trace cache (200) is further configured to periodically capture a synchronizing entry, the synchronizing entry being the address of the instruction most recently executed by the processor core (104).

7. The processor-based device of any preceding claim, wherein the trace cache (200) is further configured to provide interrupt or exception vector information.

8. The processor-based device of any preceding claim, wherein each instruction trace record further includes a data valid bit (V).

9. The processor-based device of any preceding claim, wherein the trace cache (200) is further configured to provide task identifier information.

10. The processor-based device of any preceding claim, wherein the contents of the trace cache (200) are retrievable by an operating system

11. The processor-based device of any preceding claim, wherein the communication interface comprises a serial interface (204) which is essentially compliant with the IEEE-1149.1-1990 JTAG interface standard or other similar standard.

12. The processor-based device of any preceding claim, wherein the trace cache (200) is a first-in, first-out (FIFO) circular cache.

13. The processor-based device of any of claims 1 to 12, further comprising:

a processor interface (202) coupled to the processor core (104) and the trace cache, wherein the trace cache is adapted to load the trace information from the processor core (104) via the processor interface (202), and wherein the processor core (104) is adapted to retrieve the trace information from the trace cache via the processor interface (202).

14. A method for analysing trace information in a processor-based device (102) having a processor core (104), comprising the steps of:

providing a trace cache (200) within the processor-based device (102), the trace cache comprising a series of storage elements adapted to store trace information;
capturing trace information from the processor core (104) that is indicative of the order in which the series of instructions is executed by the processor core;
storing the trace information in the trace cache storage elements as instruction trace records;
retrieving the trace information by the processor core (104) from the trace cache storage elements in response to a retrieve command; and
loading other information from the processor core (104) into the trace cache storage elements in response to a load command;
providing a communication channel (230) from the trace cache to selected pins of the processor-based device

(102); and
communicating the trace information from the trace cache (200) to the selected pins via the communication channel (230).

15. The method of claim 14, wherein the loading and retrieving steps transfer the trace information between the processor core (104) and the trace cache via a processor interface (202).

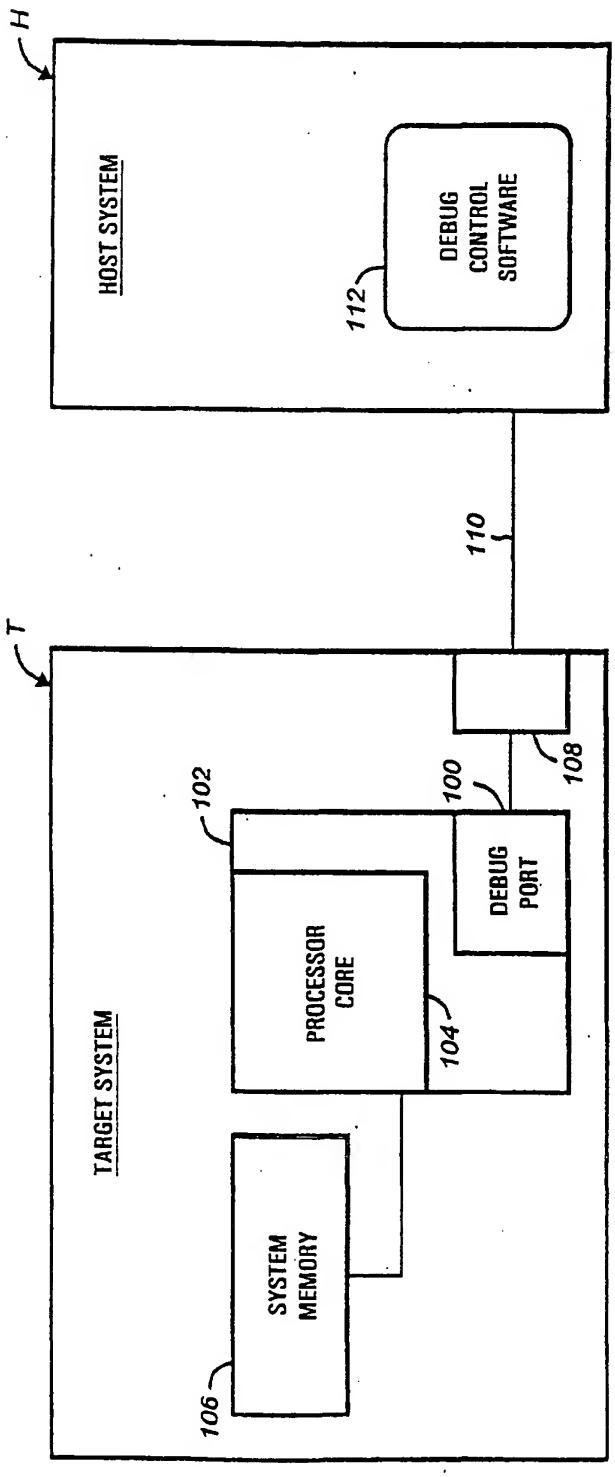

H
HOST SYSTEM
DEBUG CONTROL SOFTWARE
112
110
T
TARGET SYSTEM
102
PROCESSOR CORE
104
100
DEBUG PORT
108
SYSTEM MEMORY
106

FIG. 1

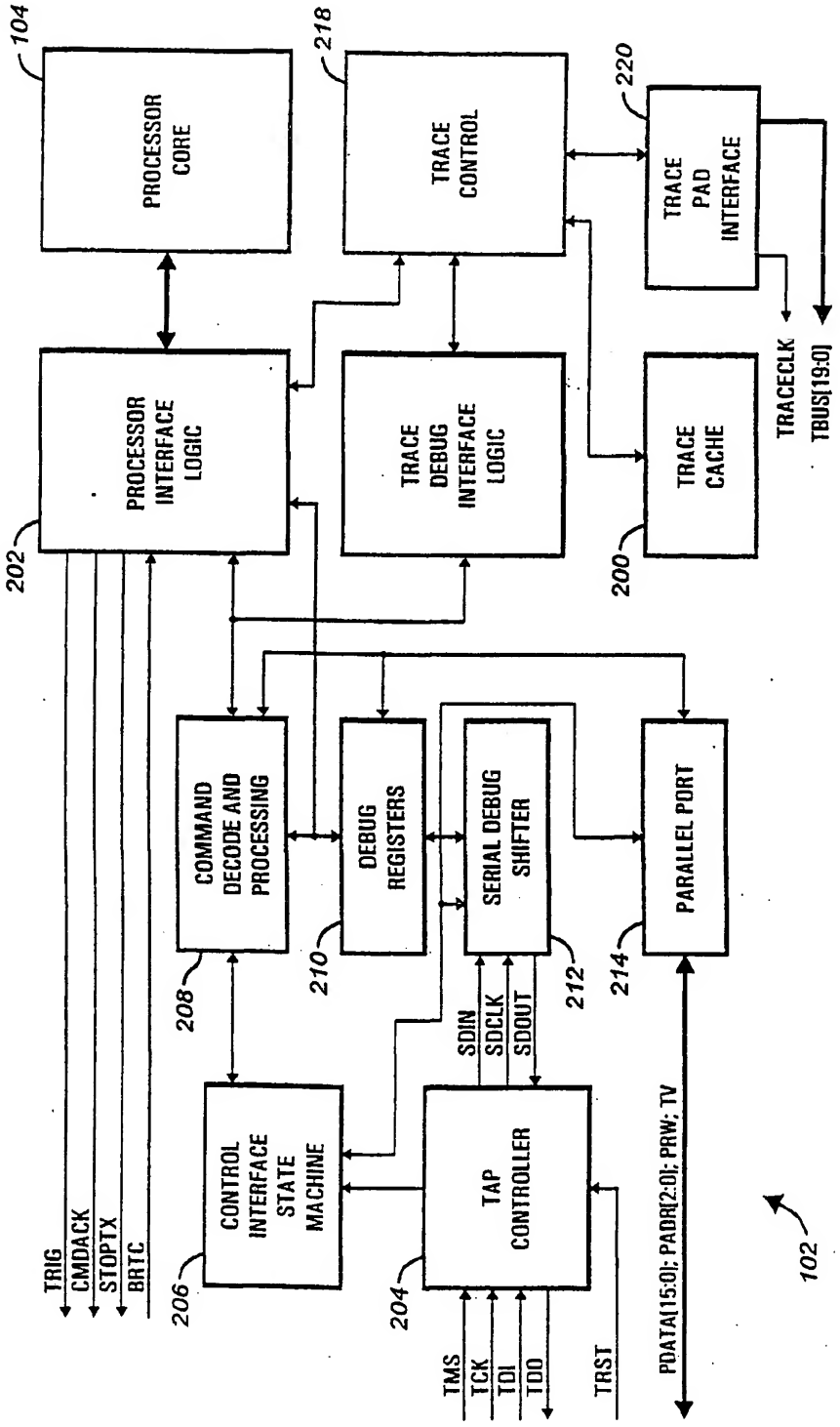

104
PROCESSOR CORE
218
TRACE CONTROL
220
TRACE PAD INTERFACE
TRACECLK
TBUS[19:0]
202
PROCESSOR INTERFACE LOGIC
TRACE DEBUG INTERFACE LOGIC
200
TRACE CACHE
COMMAND DECODE AND PROCESSING
DEBUG REGISTERS
SERIAL DEBUG SHIFTER
PARALLEL PORT
208
210
212
214
SDIN
SDCLK
SDOUT
TRIG
CMDACK
STOPTX
BRTC
CONTROL INTERFACE STATE MACHINE
TAP CONTROLLER
206
204
TMS
TCK
TDI
TDO
TRST
PDATA[15:0]; PADR[2:0]; PRW; TV
102

FIG. 2

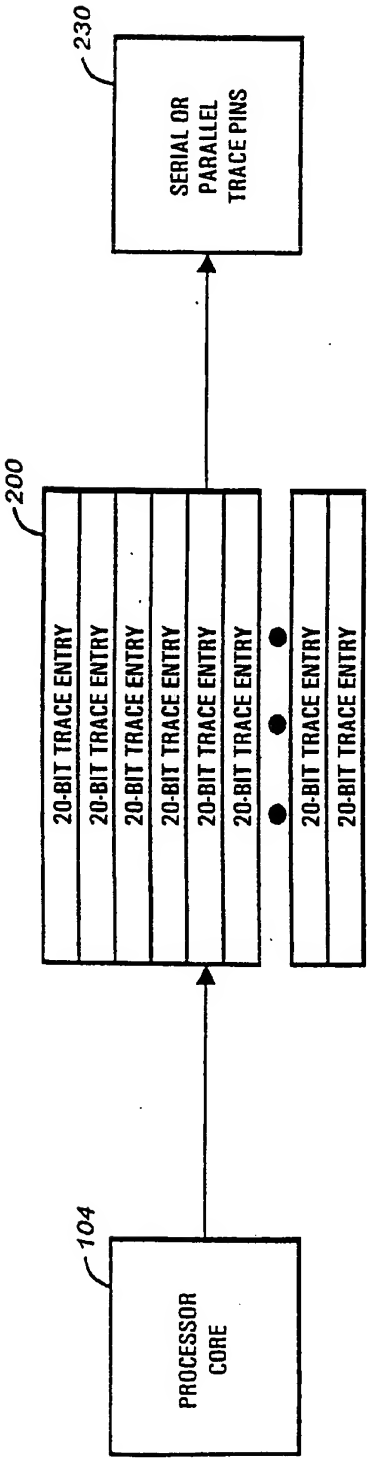

230
SERIAL OR PARALLEL TRACE PINS
200
20-BIT TRACE ENTRY
20-BIT TRACE ENTRY
20-BIT TRACE ENTRY
20-BIT TRACE ENTRY
20-BIT TRACE ENTRY
20-BIT TRACE ENTRY
20-BIT TRACE ENTRY
20-BIT TRACE ENTRY
104
PROCESSOR CORE

FIG. 3

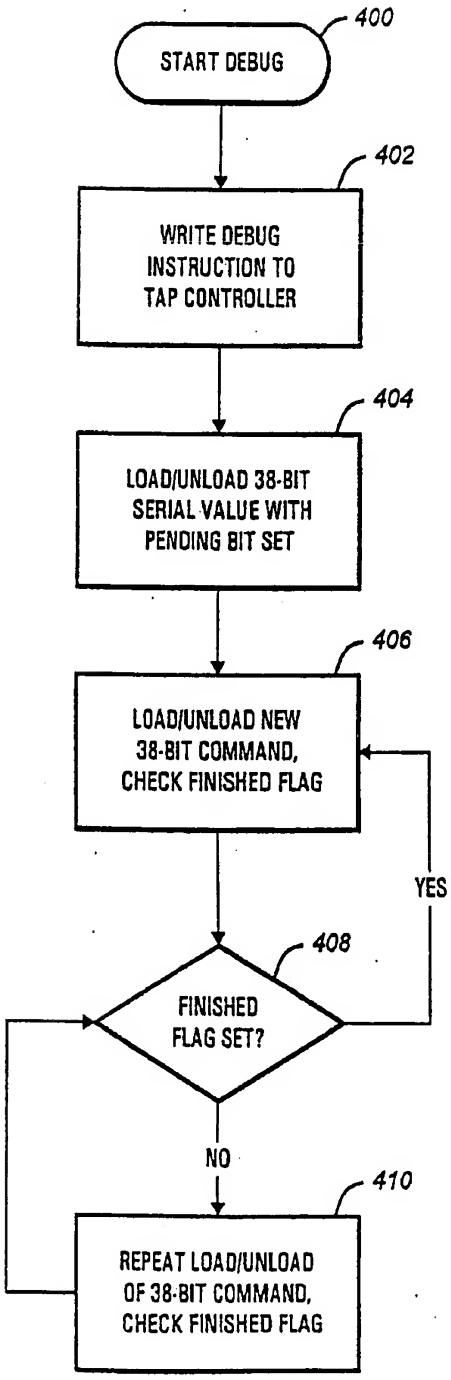

400
START DEBUG
402
WRITE DEBUG INSTRUCTION TO TAP CONTROLLER
404
LOAD/UNLOAD 38-BIT SERIAL VALUE WITH PENDING BIT SET
406
LOAD/UNLOAD NEW 38-BIT COMMAND, CHECK FINISHED FLAG
YES
408
FINISHED FLAG SET?
NO
410
REPEAT LOAD/UNLOAD OF 38-BIT COMMAND, CHECK FINISHED FLAG

FIG. 4

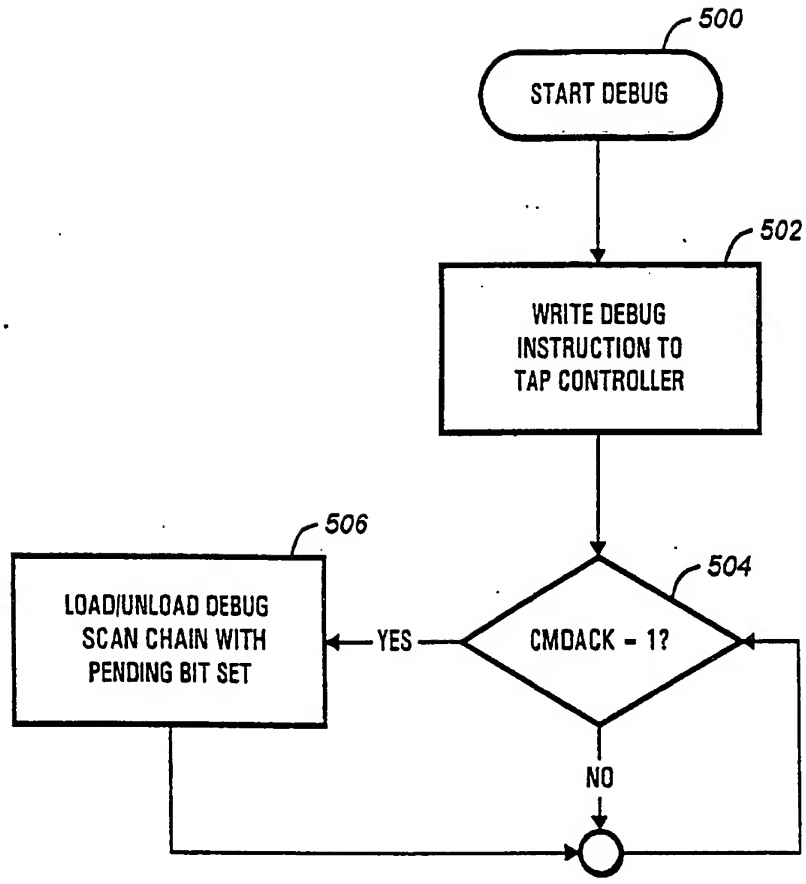

500
START DEBUG
502
WRITE DEBUG INSTRUCTION TO TAP CONTROLLER
504
CMDACK = 1?
YES
506
LOAD/UNLOAD DEBUG SCAN CHAIN WITH PENDING BIT SET
NO

FIG. 5

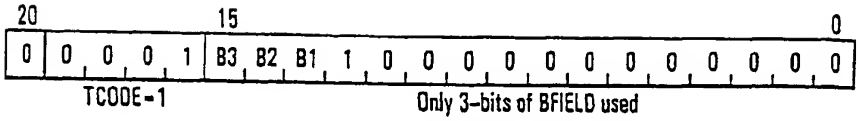


FIG. 6A

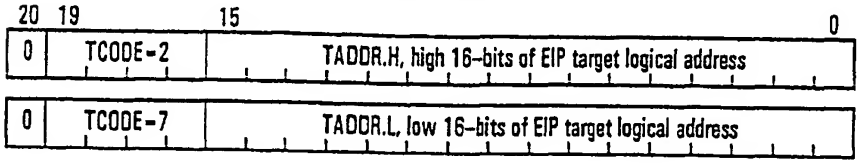


FIG. 6B

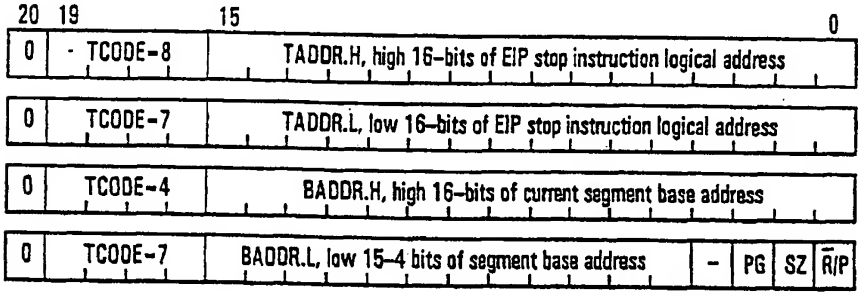


FIG. 6C

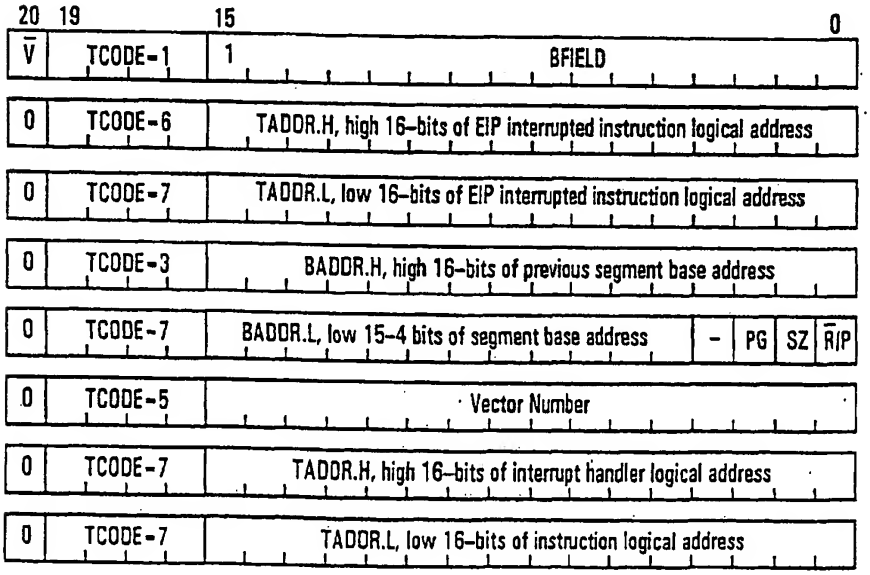


FIG. 6D

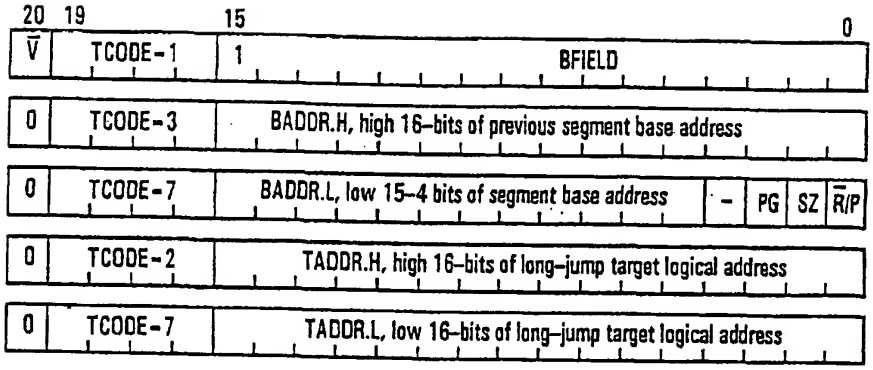


FIG. 6E

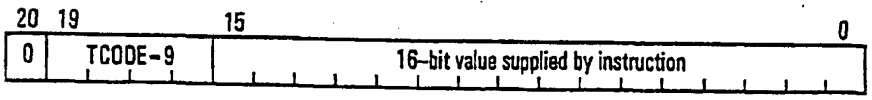


FIG. 6F

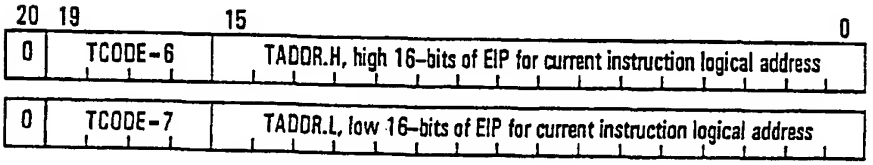


FIG. 6G